

# Cryptographic Salt: A Countermeasure against Denial-of-Service Attacks

DongGook Park<sup>1,2</sup>, JungJoon Kim<sup>1</sup>, Colin Boyd<sup>2</sup> and Ed Dawson<sup>2</sup>

<sup>1</sup>Korea Telecom, Access Network Laboratory,  
17 WooMyeon-Dong, SeoCho-Gu, 137-792, Seoul, Korea  
{dgpark6, jungkim}@kt.co.kr

<sup>2</sup>Queensland University of Technology, Information Security Research Centre,  
2 George Street, GPO Box 2434, Brisbane, Queensland 4001, Australia  
{c.boyd, e.dawson}@qut.edu.au

**Abstract.** Denial-of-service (DoS) attack is one of the most malicious Internet-based attacks. Introduction of cryptographic authentication protocols into Internet environment does not help alleviate the impact of denial-of-service attacks, but rather increases the vulnerability to the attack because of the heavy computation associated with cryptographic operation. Nevertheless, many Internet security protocols including SSL/TLS protocol do not consider this aspect. We consider this overlooked issue in authentication protocol design, and propose an effective countermeasure applicable to authentication protocols like SSL/TLS protocol which adopt public-key based encryption to authenticate the server to the client.

## 1 Introduction

Recently, DoS attack is becoming a growing concern as the Internet services have been used in more aspects of human life. Many things in human life, turned out to have their counterpart in the Internet world: DoS attack would be one example of them. In this paper, we focus on the most typical DoS attacks which may be called *connection depletion attacks* or *resource clogging attacks*: an attack in which an attacker seeks to initiate and leave unresolved a large number of connection requests to a Web server, exhausting its resources and rendering it incapable of servicing legitimate connection (or service) requests. SYN flooding attack in TCP/IP networks is the most well known example of this kind [Cert96, Fred99]. This attack exploits a weakness in the TCP connection establishment protocol. Attempting to establish a TCP connection, the client sends the server a SYN message. In response, the server sends a SYN-ACK message, and prepares the connection by allocating buffer space. The client then finishes establishing the connection by responding with an ACK message. After this sequence, both entities can exchange the service-specific data. The attacker, however, does not follow the above sequence of messages. He simply fails on purpose to send the third message, namely ACK to the server, leaving the session half-open. The attacker may initiate large amounts of SYN messages simultaneously, causing the server to be unable to handle the legitimate connection

requests. A detailed analysis of this attack and possible remedies are described by Schuba et al. [Schu97].

Using an authentication protocol in Internet environment is orthogonal to prevention of DoS attacks. Authentication protocols themselves do not help prevent denial-of-service attacks but instead may give rise to another environment for denial-of-service attacks. Usually to run an authentication protocol, the involved entity has to assign to it a particular session and some memory to keep relevant data resulting from message exchanges and related computation during the execution of it. Thus, although the notorious SYN flooding attacks can be minimized through careful design and operation of the Internet communication systems, the introduction of authentication protocols just opens up another door to similar denial-of-service attacks.

This problem concerning authentication protocols and DoS attacks is well understood and a lot of previous work is invested to address it; a detailed survey of the related work can be found in [ANL01, LNA00]. The most well studied and promising approach to date seems to be for the server to use *cookies* against a potential attacker. The concept of *cookies* for use in the context of client-server transactions started from “Netscape Cookie” in 1994 as part of the feature set of Netscape Version 1.1 [Laur98]. Since then, most Web browsers including Microsoft Explorer adopted cookies.

Cookies are pieces of information generated by a Web server and stored in the user’s computer, ready for future access [Scho99]. Basically the same concept of cookies started to be used to thwart DoS attacks on cryptographic protocols, the first example of which seems to be Photuris protocol by Karn and Simpson (most recent version 1999 [KaSi99] but originally published 1995). Several Internet security protocols followed this trend, including SKEME [Kraw96], OAKLEY [Orma98]. The basic idea of cookies in these protocols is as follows. When a client attempts to make a connection the server sends back a *cookie* which is a function of a secret known only to the server and other information unique to the particular connection. At this stage the server stores no state for this request. The client needs to return the cookie in the next message and its validity can be checked by the server from the information sent and its secret. The idea is to ensure, before investing significant resources, that the client is making a unique request for connection. This technique addresses the denial of service attacks in which the adversary sends random connection requests.

The benefits of *stateless* connections in the beginning of an authentication protocol were recognized by Janson et al. [JTY97] in the KryptoKnight protocol suite, and this concept was generalized by Aura and Nikander [AuNi97]. Their idea is to make the client store all the state information required by the server and return it to the server as necessary with each message sent. In this way the server need not store any state information. The cookie approach can be considered a special instance of the stateless connection approach in the sense that a cookie generated by the server can contain a session specific information, is stored in the client system, and later delivered back to the server to be verified.

A *cryptographic puzzle* for the client to solve to initiate a connection with the server is another approach to solving DoS attack problems. Dwork and Naor [DwNa98] first presented this concept in the context of electronic junk mailing, and later Juels and Brainard [JuBr99] presented a simpler client puzzle for the server to combat TCP SYN flooding attacks. The same concept was further developed by Aura

et al. [ANL01] to address DoS attacks against authentication protocols. In this scenario, the server in an authentication protocol can ask the client to solve a puzzle before the server creates a protocol state or computes expensive public-key related computations. In this way, the puzzle helps improve the DoS-resistance of an authentication protocol.

It can be seen that each of the countermeasures against DoS attacks that we have described carries some cost. If cookies are used as an initial stage in an authentication protocol then additional message exchanges are usually required; this can be a significant overhead in some applications such as the limited signalling channels in mobile communications. Making a protocol stateless may require significant changes to the protocol structure and also increases storage and bandwidth requirements on the client side. Finally the use of cryptographic puzzles imposes a computational burden on both client and server as well as requiring additional message exchanges.

In this paper, we propose a new countermeasure against DoS attacks for client-server security protocols in which *the client authenticates the server by sending a random nonce encrypted under the public encryption key of the server*. Such protocols include SSL/TLS [RFC99], SKEME [Kraw96], and the authentication and key agreement protocol of the PACS (Personal Access Communication System), one of the six PCS standards in North America [Bell94], [JTC94].

Our approach is on the same line of Aura and Nikander's stateless connection concept in that both approaches purport to make the cryptographic protocols themselves more robust against the attacks. Our approach has something in common with the client puzzle concept in that both use some cryptographic mechanisms to combat the DoS attacks, but differs in that our method can apply and be integrated directly into the authentication and key-establishment protocols themselves. This provides a mechanism for the design of more robust protocols. Our scheme requires only a minimal overhead on both the client and the server. The only limitation of the new method is its usage applies only to a specific type of authentication protocols as stated earlier.

### Notation

Throughout the paper, symbols  $A$  and  $B$  will denote the identities of the client and the server. Symbols like  $r_X$  denote a random number or nonce generated by principal  $X$ . The private and public keys will be written as  $K_X$  and  $K_X^{-1}$ , respectively. The encryption of some message under key  $K$  will be denoted by  $\{\bullet\}_K$  and the digital signature under key of some message under  $X$ 's private key by  $\{\bullet\}_{K_X^{-1}}$ . The hash operation of some message will be denoted by  $hash(\bullet)$  or  $H(\bullet)$ .

## 2 Server Authentication and Random Numbers

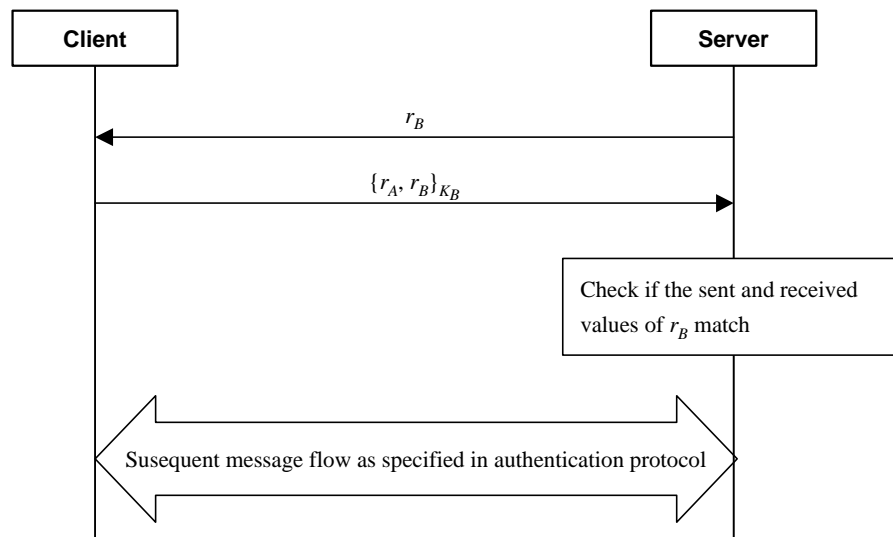
To authenticate the server with any cryptographic challenge-response mechanism, the client chooses a random number and sends it to the server. According to the way this random challenge is handled, we may have two different methods of authentication. The first is that the client can send it in the clear and then the server signs over it with its own certified private key. The corresponding public verification key is available publicly and so the client can check whether the signature was generated by and came

from the server. The unpredictability and randomness of the random challenge guarantees the required freshness of the signature: i.e., the server has generated the signature for the current session, not for another old session.

The second alternative is to encrypt the random number under the public encryption key of the server before delivery to the server. The authentic server is then the only entity to be able to retrieve the random number from the ciphertext. The server's response to the client with the decrypted random number provides the authenticity of the server's identity.

Each of the above two schemes has its own strengths and weaknesses. As far as denial-of-service attack is concerned, however, the latter method is preferable. This is because in the latter method the random number from the client is not just a random number but an encrypted message thereof, which may be exploited to accommodate a countermeasure against the DoS attack. The basic idea of the countermeasure is to implant a cryptographic salt or a random number chosen by the server in the public-key encryption operation by the client. That is, the client is required to encrypt a random nonce which he received from the server as well as his own fresh nonce. This is quite an unusual usage of random nonce encryption in public-key based authentication protocols. On receipt of the encryption message of random nonces, the server is able to check whether the message has been formed correctly since it leads to the successful retrieval of the server's random nonce after decryption *only when the message has been formed correctly*.

Figure 1 depicts this concept in more detail.



**Fig. 1.** A random number can be used as a kind of cryptographic salt to combat the DoS attack.

In the above figure, we assume that the client authenticates the server by sending the second message which is encrypted under the server's public encryption key,  $K_B$ . Furthermore, it should be noted that the first two messages just comprise a part of an

authentication and key establishment protocol, which we want to make more robust against DoS attacks. The steps in this scheme can be outlined as follows.

1. The server  $B$  chooses a random number  $r_B$  and sends it to the client  $A$ .
2. On receipt of  $r_B$ , the client chooses its own random number  $r_A$  and encrypts it together with  $r_B$  using the server's public key  $K_B$ ; the resulting ciphertext  $\{r_A, r_B\}_{K_B}$  is sent back to the server.
3. On receiving the encryption message, the server decrypts and retrieves  $r_B$  and  $r_A$  from the received ciphertext. The value of the retrieved  $r_B$  and the value of  $r_B$  which has been sent to the client should match; otherwise the server concludes that the received message is simply a garbage value sent by a malicious attacker.

Without using this kind of countermeasure, there is no way for the server to check whether the received ciphertext is really the result of a proper cryptographic computation and whether the computation has occurred for the current session. Otherwise even for a garbage or old message attack the server will execute a public key computation for decryption, send the subsequent message to the attacker, and finally will result in a state of the session left open waiting the next message from the attacker, which is simply given up by the attacker.

It can be seen that in a protocol in which the client already sends the challenge  $r_A$  encrypted using the server's public key  $K_B$  there is only a small change in the protocol messages and minimal additional computational effort required. This is in contrast to other DoS countermeasures which may require additional messages, extra computation, and/or significant alterations to the protocol specification. In practice it is often possible to include the challenge from the server in an existing message, as we will see below.

In the next section, we demonstrate that the above technique can be easily applied to a typical Internet security protocol SSL/TLS where the *ServerHello* message and the *ClientKeyExchange* message correspond to the first and the second messages, respectively.

### 3 SSL/TLS Protocol

The SSL protocol has become a de facto standard for the Internet security, and its latest version 3.0 is used as the core protocol TLS by the IETF Transport Layer Security working group. The SSL/TLS protocol uses public key cryptography for authentication and key-establishment. Some analyses of cryptographic security of the protocol have been published, such as Paulson's formal inductive analysis [Paul99], and Wagner and Schneier's informal analysis [WaSc96]. Both analyses concluded that the protocol has no weakness with regard to its basic structure. We show below its simplified abstract description which is adopted from Paulson's abstract version of the protocol, where optional messages are boxed in dotted line.

**Protocol 1.** A simplified description of the TLS handshake protocol

---

A: Client,     B: Server

1. A → B:  $A, r_A, Sid, Pa$  client hello
2. A ← B:  $r_B, Sid, BCert, Pb$  server hello, server certificate
3. A → B:  $ACert, \{r'_A\}_{K_B}, \{hash(r_B, B, r'_A)\}_{K_A^{-1}}, \{finished\}_{K_{AB}^A}$   
client certificate, client key exchange, certificate verify,  
client finished
4. A ← B:  $\{finished\}_{K_{AB}^B}$  server finished

A, B:  $M = hash(r_A, r_B, r'_A), finished = hash(Sid, M, r_A, r_B, Pa, A, Pb, B)$

---

Here we use slightly different notation from Paulson’s description of the TLS protocol;  $r_A$  and  $r_B$  replaces the original  $Na$  and  $Nb$  called client random and server random, respectively. Another random nonce  $r'_A$  denotes the pre-master-secret (*PMS*), which serves as a challenge data to the server  $B$ . The public key certificates of the client and the server are denoted as  $ACert$  and  $BCert$ , respectively.  $Sid$  means the session identifier. The notations  $\{\bullet\}_{K_B}$  and  $\{\bullet\}_{K_A^{-1}}$  stand for the message encryption under  $B$ ’s public encryption key  $K_B$  and the signature with the  $A$ ’s private signature key. Using  $r_A, r_B$  and  $M$ , the principals  $A$  and  $B$  compute the session keys  $K_{AB}^A$  and  $K_{AB}^B$  to be used for  $A$ -to- $B$  and  $B$ -to- $A$  encryptions, respectively. Whereas,  $Pa$  and  $Pb$  comply with the original notation, which mean the sets of  $A$  and  $B$ ’s preferences for encryption and compression, respectively.

We can see that  $r_B$  in the message 2 of the SSL/TLS protocol, and  $\{r'_A\}_{K_B}$  can serve a good vehicle for the countermeasure described in the previous section. That is,  $\{r'_A\}_{K_B}$  can be modified to  $\{r'_A, r_B\}_{K_B}$ . The countermeasure is very reasonable mechanism worthy to be considered for the SSL protocol because there is no additional public-key encryption/decryption required. Furthermore, it should be noted that the concatenation of  $r_A$  and  $r_B$  is not the only way to implement the idea of the countermeasure. For instance, instead of  $\{r'_A, r_B\}_{K_B}$ , we can adopt, for example, the following alternative:

$$\{r'_A + r_B\}_{K_B}, \quad hash(r'_A).$$

In this way, we can keep the length of the encrypted message as the original one. The server decrypts the received ciphertext and subtracts the value of  $r_B$  from the decrypted value and takes hash value of it, comparing it with the received value of hash. The benefit of this countermeasure can be made clearer by comparing the significance of DoS attacks for both cases: the original protocol and the modified one, as shown in the table below.

	Original SSL/TLS	Modified SSL/TLS
After a DoS attack the server has spent  and is left in a state of	<i>one</i> decryption and <i>one</i> or <i>two</i> signature verifications,  <i>one</i> half-open session.	<i>one</i> decryption,  <i>no</i> half-open session.

Here both decryption and verification are public-key based operations, and original protocol requires two signature verifications when the client authentication is needed: one for the client certificate verification and another for the client signature verification. The countermeasure cannot prevent DoS attacks completely, but significantly mitigates the damage of the attacks with no additional public key operation or extra message exchange at all.

It is very important to note that the cryptographic salt explained so far is never related to the idea of cookies. A cookie is a function of session specific information whereas a cryptographic salt is simply a nonce chosen arbitrarily by the server. Both idea, however, may be combined together as shown in the next section

#### 4 Cookies combined with the new countermeasure

The random number  $r_B$  in the countermeasure can be generated in a way similar to *cookies* in the Photuris protocol, thus enabling the server to achieve even more robustness against DoS attacks. Usually at the point of the delivery of  $r_B$  the server is expected to assign a unique session to the service requesting client. In this situation, a particular value of  $r_B$  is also uniquely related to the corresponding session. The value of  $r_B$  is stored in a memory within the server system to be compared with the received value of  $r_B$  from the client. The problem of this scheme is very similar to that of TCP/IP based client-server model which leads to the notorious SYN flooding attacks. In other words, the server must wait the second message in the above figure after it sends  $r_B$  to the client. This problem can be avoided by the server delaying the assignment of a particular session resource to the client until the client proves that he has correctly carried out the encryption of the two random nonces. In other words, the server does not couple a specific value of  $r_B$  with a particular client before the client computes and sends the required cryptographic message.

To obviate the need to store the values of  $r_B$ , the server prepares a suitable hash function  $H$ , selects a random master key  $K_{master}$  and selects a sufficiently large value as the modulus  $M$  of the index for  $r_B$ . Here, the index runs from 0 to  $M - 1$ . When a new value of  $r_B$  is required, the server runs the hash function with the master key and the current index as the inputs, the hash result of which will be used as the value of  $r_B$  (Figure 2).

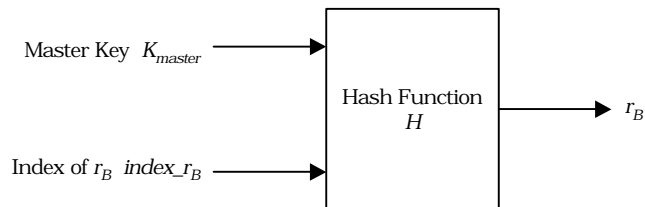
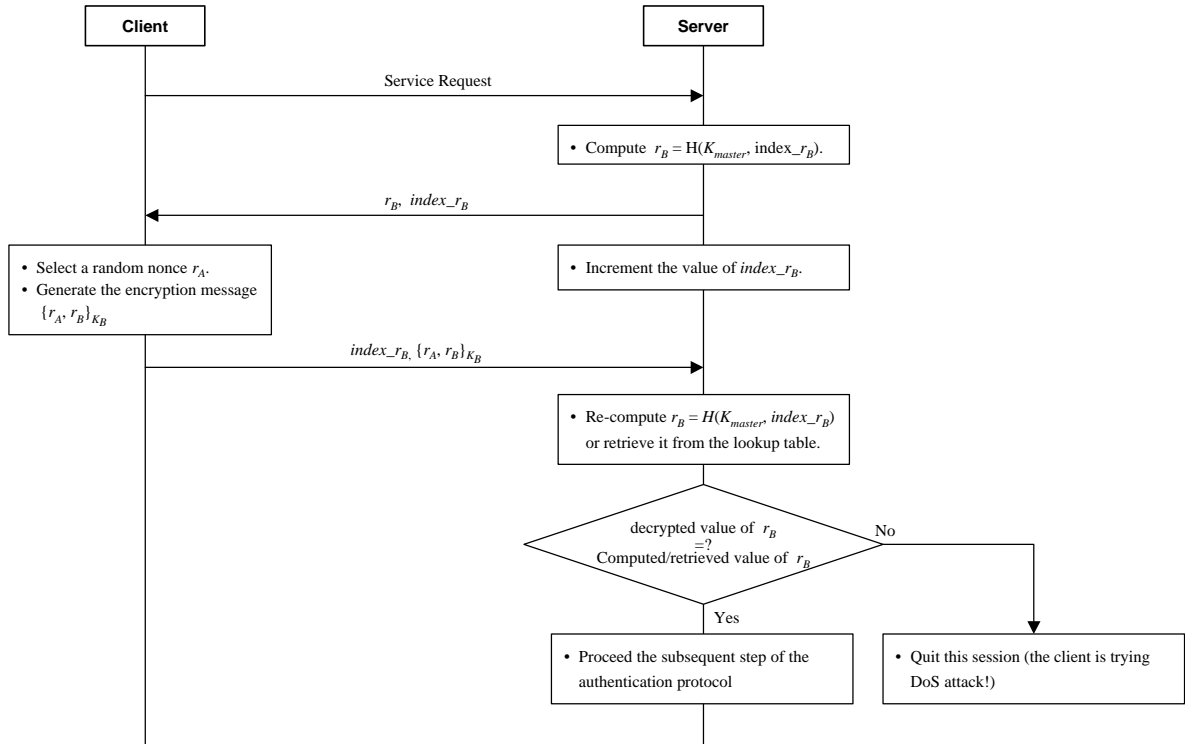


Fig. 2. Generation of random number  $r_B$

The following Figure 3 shows an example using this  $r_B$  generation method together with the countermeasure described before. The process is outlined in the following steps.



**Fig. 3.** the cryptographic salt  $r_B$  as a *cookie*

1. In response to a service request from the client, the server generates a new value of  $r_B = H(K_{master}, index\_r_B)$ , increments the index parameter  $index\_r_B$ , and sends the client the values of  $r_B$  and  $index\_r_B$ .
2. On receipt of  $r_B$  and  $index\_r_B$ , the client generates his own random nonce  $r_A$ , encrypts  $r_A$  and  $r_B$  under the public encryption key  $K_B$ , and sends the server the plaintext  $index\_r_B$  and the ciphertext  $\{r_A, r_B\}_{K_B}$ .
3. When the server receives the response from the client, using the received value of the parameter  $index\_r_B$ , it retrieves from a look-up table or, alternatively, re-computes the corresponding value of  $r_B$ . The server also decrypts the received ciphertext  $\{r_A, r_B\}_{K_B}$ , and retrieves the value of  $r_B$ , which is compared with the value of  $r_B$  which was generated by itself using the given value of  $index\_r_B$ .
4. If both values match, the server is assured that the client has formed the ciphertext honestly and sent the ciphertext  $\{r_A, r_B\}_{K_B}$ . This leads the server to the next step specified in the authentication protocol to which the protection scheme is applied.

5. On the other hand, if the match fails, the server may conclude that the client is trying DoS attack by sending a bogus message which has nothing to do with the correct cryptographic operation to compute the cipher text  $\{r_A, r_B\}_{K_B}$ .

The usage scenario of  $r_B$  as a cookie is just an example, and there may be as many usages as different uses of cookies. It should be noted, however, that the basic idea of the new countermeasure presented in this paper is rather independent from cookies. In other words, the cryptographic salt  $r_B$  as described in this paper may or may not be cookies. Rather, the new countermeasure can be more effective when combined with the cookie scheme.

## 5 Conclusion

We proposed a new concept of protecting a particular form of authentication protocols like the SSL/TLS protocol against the connection depletion attack. The cookie, an existing countermeasure, is useful against the DoS attack, but useless for a determined attacker because the cookie data can be eavesdropped by the attacker. The client puzzle approach solves the problem but requires additional computational overhead in both the client and the server. Our new concept solves all these problems without minimal overhead because it requires no additional public-key operation. In some concrete implementations of the concept, it may require one extra hash computation, which is practically insignificant. Furthermore, this protection method can be combined well with the existing cookie mechanism as well, providing more robustness against the DoS attack.

## References

- [ANL01] T. Aura, P. Nikander, J. Leiwo, "DOS-resistant authentication with client puzzles", *Proc. Security Protocols Workshop 2000*, Lecture Notes in Computer Science, Cambridge, UK, April 2000, Springer-Verlag 2001.
- [AuNi97] T. Aura and P. Nikander, "Stateless connections", *International Conference on Information and Communications Security ICICS'97*, Lecture Notes in Computer Science 1334, Springer-Verlag, 1997, pp. 87-97.
- [Bell94] TR-INS-001313, Generic Criteria for Version 0.1 Wireless Access Communications Systems (WACS), Bellcore, Revision 1, June 1994.
- [Cert96] CERT, "Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks", (Available from <http://www.cert.org/advisories/index.html>)
- [DwNa92] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail", In *Advances in Cryptology - Proc. CRYPTO '92*, volume 740 of LNCS, pages 139-147, Santa Barbara, CA USA, August 1992. Springer-Verlag.
- [Fred99] Stephen Frede, "Attack Scenarios", *Security Systems & Technologies*, September 1999, pp.4-11.
- [JTC94] JTC, Text Modification to JTC(AIR)/94.02.07-119R6, September 15, 1994.

- [JuBr99] A. Juels and J. Brainard, "Client puzzles: A cryptographic counter-measure against connection depletion attacks", *Proc. 1999 Network and Distributed System Security Symposium (NDSS)*, Internet Society, March 1999, pp. 151-165.
- [JTY97] P. Janson, G. Tsudik, and M. Yung, "Scalability and flexibility in authentication services: The KryptoKnight approach", *IEEE INFOCOM'97*, Tokyo, April 1997.
- [KaSi99] P. Karn and W. A. Simpson. Photuris: Session-key management protocol. RFC 2522, IETF Network Working Group, March 1999.
- [Kraw96] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet", *Proc. of the Internet Society Symposium on Network and Distributed System Security*, February 1996.
- [Laur98] S. St.Laurent, *cookies*, McGraw-Hill, 1998.
- [LNA00] J. Leiwo, P. Nikander, T. Aura, "Towards network denial of service resistant protocols", *Proc. Sixteenth Annual Working Conference on Information Security (SEC2000)*, IFIP Series, Vol. 175, Beijing, China, August 2000, Kluwer Academic Publishers.
- [Orma98] H. Orman. The oakley key determination protocol. *RFC2412*. The Internet Society, November 1998.
- [Paul99] L. C. Paulson, "Inductive Analysis of the Internet Protocol TLS", *ACM Transactions on Computer and System Security* 2 3, 1999, pp.332-351.
- [RFC99] T. Dierks and C. Allen, The TLS Protocol, [RFC 2246], January 1999.
- [Scho99] V. M.-Schonberger, "The Internet and Privacy Legislation: cookies for a Treat?". Available from <http://www.wvjot.wvu.edu/wvjolt/current/issue1/article>
- [Schu97] C. L. Schuba et al., "Analysis of a denial of service attack on TCP", *Proc. 1997 IEEE Symposium on Security and Privacy*, May 1997, IEEE Computer Society Press. pp. 208-223.
- [WaSc96] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol", The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, November 1996, pp.29-40.